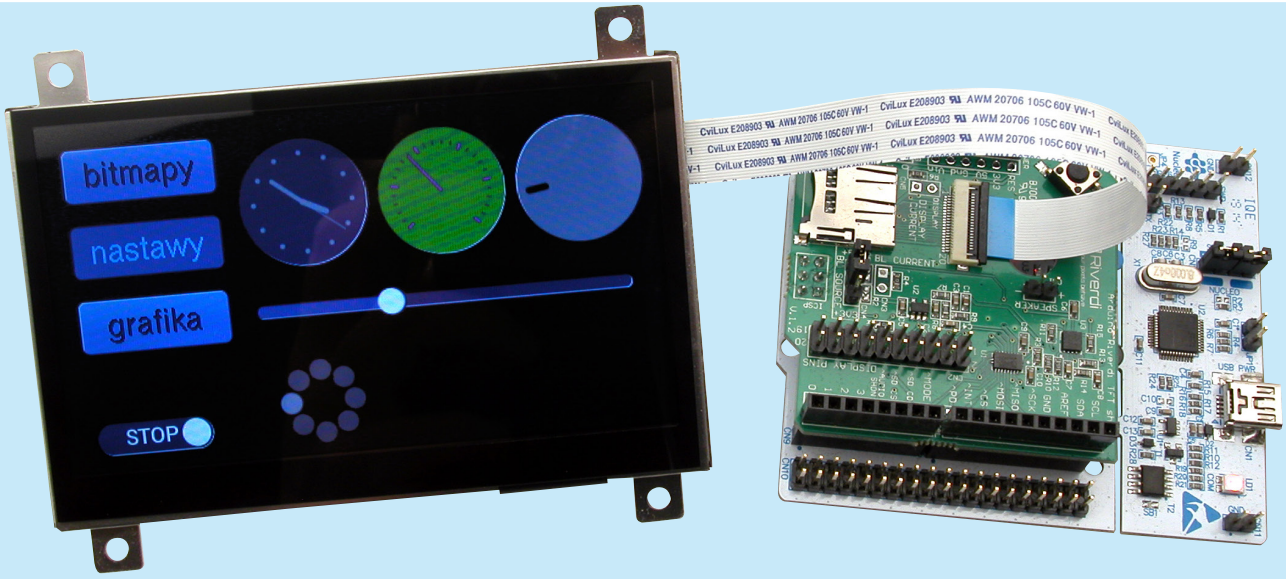


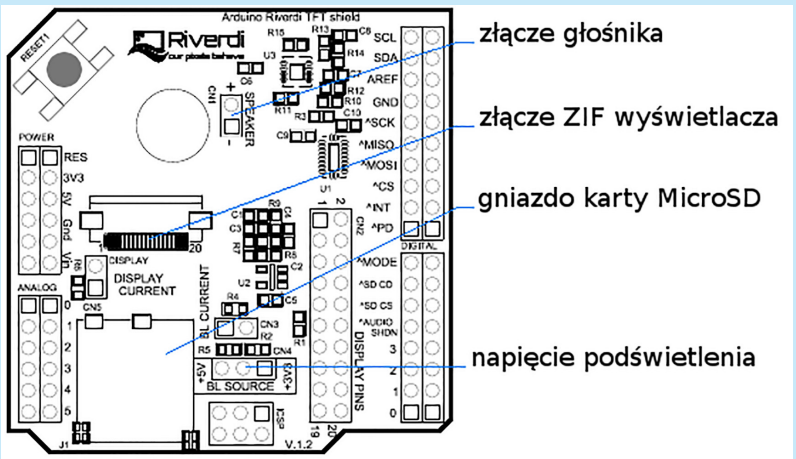
Wyświetlacz firmy Riverdi z modułem rozszerzającym dla Arduino

Dodatkowe informacje
Dziękujemy firmie Unisystem za nadesłanie zestawu firmy Riverdi.

Dołączenie kolorowego wyświetlacza TFT z pojemnościowym panelem dotykowym jako modułu rozszerzającego Arduino, tworzy możliwości budowy urządzeń z interfejsem graficznym o profesjonalnym wyglądzie i sposobie działania. W artykule – oprócz opisu wyświetlacza i modułu – sporo uwagi zostanie poświęcone procedurom programistycznym pozwalającym na uruchomienie potencjału drzemiącego w wyświetlaczu.

Opis dotyczy kolorowego wyświetlacza TFT firmy Riverdi typu RVT4.3B480272CFWC81 o wymiarach 106 mm×68 mm×9 mm, przekątnej 4,3 cala, rozdzielczości 480×272 piksele i z pojemnościowym panelem dotykowym. Możliwość montażu w obudowie ułatwia metalowa ramka z 4 uchwytami. Wyświetlacz może być sterowany za pomocą interfejsu SPI lub I²C. Do zasilania jest wymagane pojedyncze napięcie 3,3 V. Zasilanie i sygnały sterujące doprowadzane są giętką, 20-przewodową taśmą zatrzaskiwaną w rozłącznym gnieździe typu ZIF. Wyświetlacz wymaga sygnałów sterujących o napięciu z zakresu 0…3,3 V. Zamontowany układ kontrolera graficznego FT801 steruje wyświetlaczem, panelem dotykowym oraz odpowiada za obsługę funkcji graficznych i efektów dźwiękowych.

Moduł rozszerzający o nazwie „Arduino Riverdi TFT Shield" umożliwia dołączenie i zastosowanie wspomnianego wyświetlacza w projektach opartych o płytki Arduino oraz z nimi zgodne pod względem wymiarów i rozmieszczenia wyprowadzeń. Oprócz typowych dla standardu złącz, na płytce modułu zamontowano gniazdo ZIF dla taśmy wyświetlacza, gniazdo dla karty MicroSD, monofoniczny wzmacniacz akustyczny TPA6205A, złącze głośnika i przycisk *Reset*.



Rysunek 1. Rozmieszczenie złączy na płytce adaptera wyświetlacza

Gniazda i złącza modułu rozszerzającego

Na **rysunku 1** pokazano rozmieszczenie elementów zamontowanych na płytce modułu adaptera. Najważniejsze są złącza umożliwiające połączenie modułu z innymi płytkami

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

Tabela 1. Sygnały wyprowadzone na gniazdo CN2

Styk	Złącze	Złącze Arduino	Uwagi
1	VDD	3V3	Zasilanie wyświetlacza 3,3 V/70 mA
2	GND	GND	Masa
3	SPI_SCLK/ I2C_SCL	SCK	SPI sygnał SCLK lub I²C sygnał SCL, wewnętrznie podciągany przez 47 k do VDD
4	SPI_MISO/ I2C_SDA	MISO	SPI sygnał MISO lub I²C sygnał SDA, wewnętrznie podciągany przez 47 k do VDD
5	SPI_MOSI/ I2C_SA0	MOSI	SPI sygnał MOSI lub I²C adres slave b.0, wewnętrznie podciągany przez 47 k do VDD
6	SPI_CS/ I2C_SA1	D10	SPI sygnał CS lub I2C adres slave b.1, wewnętrznie podciągany przez 47 k do VDD
7	INT	D9	Wyjście sygnału przerwania, aktywny poziom niski, wewnętrznie podciągany przez 47 k do VDD
8	PD	D8	Wejście sygnału Power Down, aktywny poziom niski, wewnętrznie podciągany przez 47 k do VDD
9	MODE	D7	Wejście wyboru interfejsu: SPI (poziom niski), I²C (poziom wysoki) wewnętrznie podciągany przez 10 k do VDD
10	AUDIO_OUT	–	Wyjście sygnału audio
11-16	n.c.	–	Nie podłączać
17-18	BLVDD	–	Zasilanie podświetlenia 3,3 V/260 mA lub 5 V/150 mA
19-20	BLGND	–	Masa podświetlenia

Arduino. Oprócz tego, na rysunku zaznaczono położenie takich elementów, jak: gniazdo ZIF dla taśmy łączącej płytkę z wyświetlaczem, położenie gniazda dla karty MicroSD, złącza dla dołączenia zewnętrznego głośnika, zwory wyboru napięcia podświetlania: (5 V lub 3,3 V).

Sygnały ze złącza ZIF wyświetlacza są wyprowadzone na szpilki gniazda CN2. Dzięki temu można podłączyć się bezpośrednio do wyświetlacza i sterować nim z zewnętrznego kontrolera. W **tabeli 1** umieszczono wykaz wszystkich sygnałów wyprowadzonych na gniazdo CN2.

Wybór interfejsu do komunikacji z wyświetlaczem

Do komunikacji z wyświetlaczem można użyć interfejsu I²C lub SPI. O wyborze decyduje poziom sygnału MODE na wyprowadzeniu 9 złącza wyświetlacza. Poziom niski oznacza wybór interfejsu SPI, a wtedy na linie 3…6 wyprowadzone są sygnały SCLK, MISO, MOSI, CS. Poziom wysoki sygnału powoduje wybranie interfejsu I²C, a wtedy na linie 3…6 wyprowadzone są sygnały SCL, SDA, bity „A0” i „A1” adresu. Wyświetlacz reaguje na 7-bitowy adres I²C z przedziału 20…23h ustalany za pomocą I2C_SA0 i I2C_SA1.

Sterownik FT801

Pracą wyświetlacza steruje układ kontrolera FT801. Odpowiada on za wytwarzanie sygnałów sterujących pracą matrycy TFT oraz za operacje graficzne i dźwiękowe. W tym – za podstawowe operacje rysowania punktów, linii, figur geometrycznych, generowanie obrazu złożonych komponentów, takich jak: przyciski, suwaki, przełączniki, wskaźniki i inne. Sterowanie FT801 dokonuje się poprzez odwołanie do określonych obszarów adresowych pamięci układu i rozkazy sterujące.

Informacje techniczne dotyczące samego układu FT801 oraz sposobów jego sterowania można znaleźć na stronie firmy FTDI zamieszczonej pod adresem *http://goo.gl/oli1Jb*. W dokumencie *DS_FT801.pdf* są zawarte dane techniczne oraz opis budowy sterownika. W dokumencie *FT800 Programmers Guide.pdf* można znaleźć opis programowania, adresów i komend pozwalających na sterowanie układem i wygenerowanie na ekranie obrazów komponentów. Pod podanym adresem można

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

Tabela 2. Połączenia pomiędzy mikrokontrolerem a wyświetlaczem		
Port kontrolera	**Sygnał wyświetlacza**	**Uwagi**
Interfejs SPI1: PA5, PA6, PA7	SPI_SCLK, SPI_MISO, SPI_MOSI	Interfejs SPI i porty interfejsu wykorzystywane do komunikacji z wyświetlaczem. Tryb Full_Duplex_Master
PA8	MODE	Linia MODE wyświetlacza. Gdy =0 aktywny interfejs SPI
PA9	PD	Sterowanie obniżonym poborem mocy. Gdy PD =1 normalna praca wyświetlacza
PB6	SPI_CS	Gdy SPI_CS =0 zostaje otwarta komunikacja interfejsem SPI

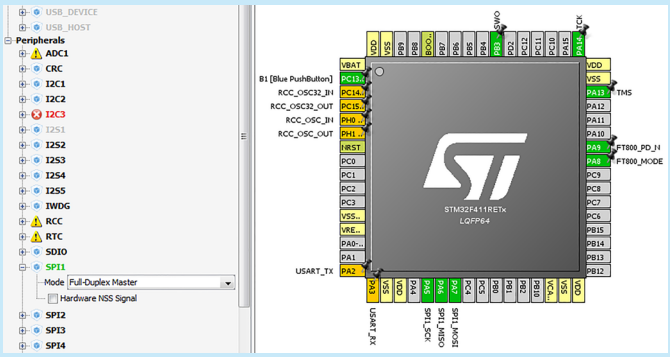
także znaleźć pliki z przykładami programów dla platformy Arduino, mikrokontrolerów ARM oraz PIC. Są tam także linki do pobrania narzędzi ułatwiających pisanie oprogramowania dla FT801.

Przykładowy program demo i biblioteki

Dla ułatwienia zrozumienia sposobu programowania FT801, opracowano program demonstracyjny pokazujący niektóre z możliwości układu. Pliki źródłowe programu zostały wygenerowane przez pakiet programistyczny *Ac6 System Workbench for STM32 – C/C++ Embedded Development Tools*. Jako część sprzętową przy testowaniu oprogramowania wykorzystany został wyświetlacz RVT4.3B480272CFWC81, moduł rozszerzający „Arduino Riverdi TFT Shield" oraz płytka stm32NUCLEO-F411RE. Program demonstracyjny i pliki bibliotek oparte zostały na przykładzie udostępniony przez firmę FTDI dla kontrolera ARM do pobrania ze strony *http://goo.gl/x0txGI*.

W skład programu demonstracyjnego wchodzą osobne pliki biblioteki procedur dla FT801. Dzięki temu mogą być łatwo przeniesione do oprogramowaniu pisanego dla innego kontrolera i pakietu programistycznego. Oprogramowanie przykładowe korzysta z biblioteki HAL dla kontrolera STM32F411. Komunikacja z układem FT801 odbywa się za pośrednictwem interfejsu SPI. Struktura plików jest następująca:

- **FT801_procedury.c** i **FT801_procedury.h** (pliki zawierające procedury inicjujące FT801 i podstawowe procedury transmisji rozkazów i danych do układu).
- **FT801_funkcje.c** i **FT801_funkcje.h** (biblioteka procedur graficznych do rysowania figur geometrycznych, wyświetlania obrazów, tekstów oraz złożonych komponentów graficznych, jak przyciski, suwaki, wskaźniki itp.).
- **FT800.h** (plik nagłówkowy z deklaracjami stałych wykorzystywanych przez bibliotekę).



Rysunek 2. Obraz wyświetlany w oknie STM32CubeMX po poprawnym skonfigurowaniu połączeń

- **Procedury_Demo_FT801.c** i **Procedury_Demo_FT801.h** (plik główny programu demonstracyjnego).
- **Demo_Obrazy.c** i **Demo_Obrazy.h** (pliki przykładowych bitmap).

Wygenerowanie szkieletu oprogramowania

Wyświetlacz jest sterowany przez płytkę STM32NUCLEO-F411RE. Do wygenerowania szkieletu oprogramowania użyto programu *STM32CubeMX*. Za jego pomocą skonfigurowano interfejsy i wyprowadzenia, które zostały użyte do komunikacji pomiędzy płytką NUCLEO a wyświetlaczem. Połączenia pomiędzy mikrokontrolerem a wyświetlaczem wymieniono w **tabeli 2**.

Na **rysunku 2** pokazano obraz wyświetlany w oknie *STM32CubeMX* po poprawnym skonfigurowaniu połączeń. Po zaznaczeniu w opcjach *Project → Settings* nazwy projektu i jego folderu na dysku, wskazano wykorzystywane IDE – w tym wypadku SW4STM32. Po wybraniu *Project → Generate Code* został automatycznie wygenerowany i zapisany we wskazanym miejscu szkielet oprogramowania. Tak przygotowane

Listing 1. Przykład użycia procedury przesyłającej do wyświetlacza 32 bity danych za pomocą SPI

```
void ft800memWrite32(unsigned long ftAddress, unsigned long ftData32)
{
    unsigned char cTempAddr[3];  // adres pamięci FT800
    unsigned char cTempData[4];     // bufor słów 32-bitowych
    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_WRITE;
    cTempAddr[1] = (char) (ftAddress >> 8);
    cTempAddr[0] = (char) (ftAddress);
    cTempData[3] = (char) (ftData32 >> 24);
    cTempData[2] = (char) (ftData32 >> 16);
    cTempData[1] = (char) (ftData32 >> 8);
    cTempData[0] = (char) (ftData32);
    HAL_GPIO_WritePin(GPIOB, FT800_CS_N, GPIO_PIN_RESET);  //CS=0
    for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi1, &cTempAddr[i], 1, 0);  //komenda Memory Write+MSB adresu
    }
    for (int j = 0; j < sizeof(cTempData); j++)  //początek od LSB adresu
    {
        HAL_SPI_Transmit(&hspi1, &cTempData[j], 1, 0);  //wysłanie bajtu przez SPI
    }
    HAL_GPIO_WritePin(GPIOB, FT800_CS_N, GPIO_PIN_SET);  //CS=1
}
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

Listing 2. Przykład użycia procedury odczytującej z wyświetlacza 32 bity danych za pomocą SPI

```
unsigned long ft800memRead32(unsigned long ftAddress)
{
    unsigned long ftData32; //liczba pobierana z FT800
    unsigned char cTempAddr[3];    //adres w pamięci FT800
    unsigned char cTempData[4];    //bufor odczytu
    unsigned char cZeroFill;

    cTempAddr[2] = (char) (ftAddress >> 16) | MEM_READ; //komenda do wysłania
    cTempAddr[1] = (char) (ftAddress >> 8);
    cTempAddr[0] = (char) (ftAddress);
    HAL_GPIO_WritePin(GPIOB, FT800_CS_N, GPIO_PIN_RESET); //CS=0
    for (int i = 2; i >= 0; i--)
    {
        HAL_SPI_Transmit(&hspi1, &cTempAddr[i], 1, 0);  //komenda Memory Write+MSB adresu
    }
    HAL_SPI_Transmit(&hspi1, &cZeroFill, 1, 0);
    for (int j = 0; j < sizeof(cTempData); j++)
    {
        HAL_SPI_Receive(&hspi1, &cTempData[j], 1, 0);
    }
    HAL_GPIO_WritePin(GPIOB, FT800_CS_N, GPIO_PIN_SET); //CS=1
    ftData32 = (cTempData[3]<< 24) | //wartość zwracana przez funkcję
        (cTempData[2]<< 16) |
        (cTempData[1]<< 8) |
        (cTempData[0]);
    return ftData32;
}
```

Listing 3. Procedura kontrolująca stan zapełnienia bufora kołowego

```
unsigned int incCMDOffset(unsigned int currentOffset, unsigned char commandSize)
{
    unsigned int newOffset;
    newOffset = currentOffset + commandSize;
    if(newOffset > 4095)
    {
        newOffset = (newOffset - 4096);
    }
    return newOffset;
}
```

Listing 4. Deklaracja wartości zmiennych odpowiadających użytej matrycy TFT

```
lcdWidth   = 480;  // Active width of LCD display
lcdHeight  = 272;  // Active height of LCD display
lcdHcycle  = 548;  // Total number of clocks per line
lcdHoffset  = 43;  // Start of active line
lcdHsync0 = 0;     // Start of horizontal sync pulse
lcdHsync1  = 41;   // End of horizontal sync pulse
lcdVcycle  = 292;  // Total number of lines per screen
lcdVoffset = 12;   // Start of active screen
lcdVsync0  = 0;    // Start of vertical sync pulse
lcdVsync1  = 10;   // End of vertical sync pulse
lcdPclk    = 5;    // Pixel Clock
lcdSwizzle = 0;    // Define RGB output pins
lcdPclkpol = 1;    // Define active edge of PCLK
```

Listing 5. Inicjowanie interfejsu SPI i jego wyprowadzeń

```
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);  //MODE=0 aktywny interfejs SPI
HAL_GPIO_WritePin(GPIOA, FT800_PD_N, GPIO_PIN_SET);    // Initial state of PD_N - high
HAL_GPIO_WritePin(GPIOB, FT800_CS_N, GPIO_PIN_SET);    // Initial state of SPI CS - high
HAL_Delay(20);
HAL_GPIO_WritePin(GPIOA, FT800_PD_N, GPIO_PIN_RESET);  // Reset FT800
HAL_Delay(20);
HAL_GPIO_WritePin(GPIOA, FT800_PD_N, GPIO_PIN_SET);    // FT800 is awake
HAL_Delay(20);
ft800cmdWrite(FT800_ACTIVE);  // Start FT800/FT801
HAL_Delay(5);
ft800cmdWrite(FT800_CLKEXT);  // Set FT800 for external clock
HAL_Delay(5);
ft800cmdWrite(FT800_CLK48M);  // Set FT800 for 48MHz PLL
HAL_Delay(5);
// Now FT800 can accept commands at up to 30 MHz clock on SPI bus
```

pliki szkieletu nadają się do wczytania przez środowisko programistyczne *Ac6 System Workbench for STM32 – C/C+*.

Procedury inicjujące

Wygenerowany przez *STM32CubeMX* szkielet oprogramowania zawiera procedury inicjujące interfejs SPI i porty sterujące wyświetlaczem. Należy je uzupełnić procedurami transmisji i odbioru wykorzystującymi interfejs SPI oraz procedurą przygotowującą wyświetlacz do odbioru i wykonywania rozkazów sterujących. Wszystkie znajdują się w pliku *FT801_procedury.c* programu demonstracyjnego.

Procedury transmisji korzystają ze sprzętowego interfejsu SPI wbudowanego w kontroler STM32F411. Jeżeli biblioteki miały by być wykorzystywane dla innego typu kontrolera, należy zmienić tylko te fragmenty kodu, pozostałe nie będą wymagały zmian. Procedury transmisji i odbioru zostały opracowane dla obsługi danych 8-, 16-, i 32-bitowych.

Na **listingu 1** pokazano przykład użycia procedury przesyłającej do wyświetlacza 32 bity danych za pomocą SPI. Procedura jest wywoływana z dwoma argumentami: adresem i 32-bitową daną do zapisu. W procedurze użyto biblioteki HAL dla mikrokontrolera STM32F411. Analogiczną procedurę odczytu danej 32-bitowej spod wskazanego adresu pamięci kontrolera FT801 zamieszczono na **listingu 2**.

Ponieważ rozkazy dla sterownika FT801 wpisywane są najpierw do bufora kołowego w wewnętrznej pamięci układu, przydatna będzie procedura kontrolująca stan jego zapełnienia – zaprezentowano ją na **listingu 3**. Jest

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

Listing 6. Zawartość pliku ft800.h

```
if (ft800memRead8(REG_ID) != 0x7C) // Read ID register - is it 0x7C?
{
    while(1);    // If we don't get 0x7C, the ineface isn't working - halt with infinite loop
}
ft800memWrite8(REG_PCLK, ZERO);  // Set PCLK to zero - don't clock the LCD until later
ft800memWrite8(REG_PWM_DUTY, ZERO);  // Turn off backlight
// End of Wake-up FT800
```

Listing 7. Zapisanie stałych do rejestrów FT801 oraz wyłączenie audio

```
ft800memWrite16(REG_HSIZE, lcdWidth);  // active display width
ft800memWrite16(REG_HCYCLE, lcdHcycle);  // total number of clocks per line, incl front/back porch
ft800memWrite16(REG_HOFFSET, lcdHoffset);  // start of active line
ft800memWrite16(REG_HSYNC0, lcdHsync0);  // start of horizontal sync pulse
ft800memWrite16(REG_HSYNC1, lcdHsync1);  // end of horizontal sync pulse
ft800memWrite16(REG_VSIZE, lcdHeight);  // active display height
ft800memWrite16(REG_VCYCLE, lcdVcycle);  // total number of lines per screen, incl pre/post
ft800memWrite16(REG_VOFFSET, lcdVoffset);  // start of active screen
ft800memWrite16(REG_VSYNC0, lcdVsync0);  // start of vertical sync pulse
ft800memWrite16(REG_VSYNC1, lcdVsync1);  // end of vertical sync pulse
ft800memWrite8(REG_SWIZZLE, lcdSwizzle);  // FT800 output to LCD - pin order
ft800memWrite8(REG_PCLK_POL, lcdPclkpol);  // LCD data is clocked in on this PCLK edge
ft800memWrite8(REG_VOL_PB, ZERO);  // turn recorded audio volume down
ft800memWrite8(REG_VOL_SOUND, ZERO);  // turn synthesizer volume down
ft800memWrite16(REG_SOUND, 0x6000);  // set synthesizer to mute
```

Lisitng 8. Sprawdzenie bufora kołowego oraz operacje na wyświetlaczu

```
ramDisplayList = RAM_DL;  // start of Display List
ft800memWrite32(ramDisplayList, DL_CLEAR_RGB);  // Clear Color RGB   00000010 RRRRRRRR GGGGGGGG
BBBBBBBB
//(R/G/B = Colour values) default zero / black
ramDisplayList += 4;  // point to next location
ft800memWrite32(ramDisplayList, (DL_CLEAR | CLR_COL | CLR_STN | CLR_TAG));  // Clear 00100110 -----
--- --------
//-----CST  (C/S/T define which parameters to clear)
ramDisplayList += 4;  // point to next location
ft800memWrite32(ramDisplayList, DL_DISPLAY);  // DISPLAY command 00000000 00000000 00000000
00000000 (end
//of display list)
ft800memWrite32(REG_DLSWAP, DLSWAP_FRAME);  // 00000000 00000000 00000000 000000SS  (SS bits define
//when render occurs)
// Nothing is being displayed yet... the pixel clock is still 0x00
ramDisplayList = RAM_DL;  // Reset Display List pointer for next list
ft800Gpio = ft800memRead8(REG_GPIO);  // Read the FT800 GPIO register for a read/modify/write
operation
ft800Gpio = ft800Gpio | 0x80;  // set bit 7 of FT800 GPIO register (DISP) - others are inputs
ft800memWrite8(REG_GPIO, ft800Gpio);  // Enable the DISP signal to the LCD panel
ft800memWrite8(REG_PCLK, lcdPclk);  // Now start clocking data to the LCD panel
for(int duty = 0; duty <= 128; duty++)
{
    ft800memWrite8(REG_PWM_DUTY, duty);  // Turn on backlight - ramp up slowly to full brighness
    HAL_Delay(10);
}
```

Listing 9. Struktura i funkcja wyświetlające tekst na ekranie

```
typedef struct S_CMD_TXT
{
    int16_t x;  //poz.x tekstu
    int16_t y;  //poz.y tekstu
    int16_t font;  //font 0-31
    uint16_t options;  //opcje wyświetlania
    const char* s;  //wskaźnik do wyświetlanego tekstu
}s_cmd_txt;

//funkcja wyświetlania tekstu
//we:    *cmdOffset        -wskaźnik do rejestru przesunięcia pozycji zapisu
//       s_cmd_txt         -struktura zawierająca parametry funkcji
void FT80x_F_CMD_Text(unsigned int *p_cmdOffset, s_cmd_txt cmd_txt)
{
unsigned int cmdOffset;
    cmdOffset =*p_cmdOffset;
    ft800memWrite32(RAM_CMD + cmdOffset, CMD_TEXT);
    cmdOffset = incCMDOffset(cmdOffset, 4);
    ft800memWrite16(RAM_CMD + cmdOffset, cmd_txt.x);
    cmdOffset = incCMDOffset(cmdOffset, 2);
    ft800memWrite16(RAM_CMD + cmdOffset, cmd_txt.y);
    cmdOffset = incCMDOffset(cmdOffset, 2);
    ft800memWrite16(RAM_CMD + cmdOffset, cmd_txt.font);
    cmdOffset = incCMDOffset(cmdOffset, 2);
    ft800memWrite16(RAM_CMD + cmdOffset, cmd_txt.options);
    cmdOffset = incCMDOffset(cmdOffset, 2);
    while (1)
    {
        ft800memWrite8(RAM_CMD + cmdOffset, *cmd_txt.s);
        cmdOffset = incCMDOffset(cmdOffset, 1);
        if (*cmd_txt.s ==’\0’) break;
        cmd_txt.s++;
    }
    while ((cmdOffset %4) !=0)
    {
        ft800memWrite8(RAM_CMD + cmdOffset, ‘\0’);
        cmdOffset = incCMDOffset(cmdOffset, 1);
    }
    *p_cmdOffset =cmdOffset;
}
```

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 11875, pass: 6hhcxxtt

Lisiting 10. Przykład użycia procedury do wyświetlenia tekstu o foncie 24 i znakach w kolorze zielonym

```
FT80x_F_DL_Color_RGB(&cmdOffset, 0, 255, 0);
cmd_txt.x =311;
cmd_txt.y =78;
cmd_txt.font =24;
cmd_txt.options =OPT_CENTER;
cmd_txt.s ="RVT4.3B480272CFWC81";
FT80x_F_CMD_Text(&cmdOffset, cmd_txt);
```

ona wywoływana z 2 parametrami: aktualną pozycją do zapisu i liczbą bajtów zapisywanego rozkazu. Zwracana jest nowa, aktualna pozycja do kolejnego zapisu.

Procedura inicjująca wyświetlacz po włączeniu zasilania rozpoczyna się od deklaracji wartości zmiennych odpowiadających użytej matrycy TFT – pokazano je na **listingu 4**. Następnie, ustawiane są odpowiednie poziomy na liniach portów kontrolujących sygnały MODE, PD i CS wyświetlacza oraz konfigurowane rejestry FT801 odpowiadające za tryb pracy SPI (**listing 5**).

Jeżeli teraz wartość odczytana spod adresu REG_ID będzie równa 0x7C, to oznacza nawiązanie komunikacji z kontrolerem FT801. Adres REG_ID oraz inne stałe używane w procedurach są zadeklarowane w w pliku *FT800.h* zamieszczonym na **listingu 6**. W kolejnym kroku, do rejestrów FT801 zapisuje się stałe zależne od użytej matrycy TFT i można wyłączyć układ audio (**listing 7**). Na koniec są wykonywane operacje związane z buforem kołowym FT801, do którego będą przesyłane rozkazy – pokazano je na **listingu 8**.

Po zakończeniu prawidłowej procedury inicjującej ekran wyświetlacza powinien być jednolity, ciemny z załączonym podświetleniem tła. Sterownik wyświetlacza FT801 jest gotowy do przyjmowania rozkazów graficznych.

Funkcje biblioteki procedur graficznych FT801

Biblioteka użyta w programie demonstracyjnym składa się z szeregu procedur przeznaczonych do wykonania określonej czynności, np. wyświetlenia przycisku. Przed wywołaniem procedury należy zapisać parametry wyświetlanego obiektu do skojarzonej z procedurą struktury. Struktury i procedury są umieszczone w plikach *FT801_funkcje.c* i *FT801_funkcje.h*. Struktura i funkcja pozwalająca wyświetlić tekst na ekranie wyświetlacza wyglądają następująco: listing 9.

Przykład użycia procedury do wyświetlenia tekstu o foncie 24 i znakach w zielonym kolorze pokazano na **listingu 10**.

Programy narzędziowe do projektowania ekranów

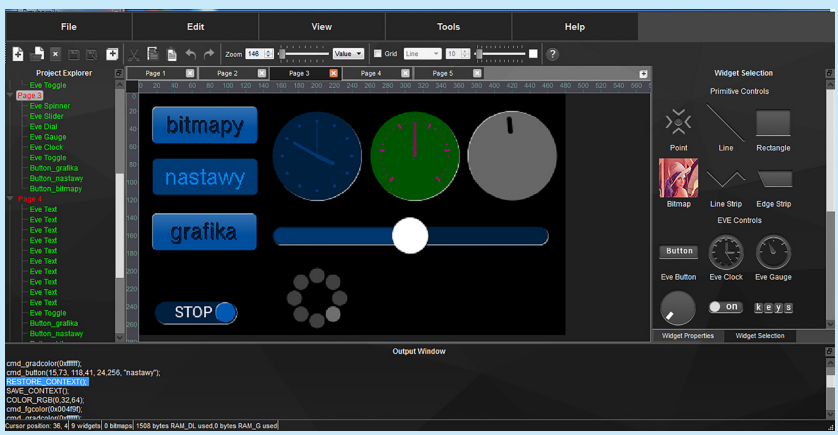
Projektując samodzielnie wygląd ekranu, który ma być wyświetlony na wyświetlaczu, trzeba wyliczyć pewne parametry, takie jak pozycje i wielkość komponentów na ekranie. Dla ułatwienia życia użytkownikom sterowników FT800 i FT801, firma FTDI opracowała narzędzia wspomagające projektowanie ekranu i tworzenie kodu rozkazów dla sterownika. Są to dwa programy: **FTDI EVE Screen Editor** i **FTDI EVE Screen Designer**.

Pliki instalacyjne programów można pobrać ze strony *http://goo.gl/mqq5tW*. Korzystając z wymienionych programów można metodą przesuwania komponentów ustalić optymalny wygląd wyświetlanego ekranu, dobrać wielkość, kształt, kolor komponentów a nawet wygenerować szkielet programu zawierający wywołania procedur i rozkazów.

Screen Designer wydaje się łatwiejszy w użyciu, widok jego pulpitu pokazano na **rysunku 3**. Z prawej strony pulpitu znajduje się przybornik z dostępnymi komponentami. Po kliknięciu na komponent i przytrzymaniu przycisku myszy można przesunąć komponent w wybrane miejsce na środkowej części pulpitu, symbolizującej ekran wyświetlacza. Korzystając z uchwytów zaznaczonych na ramce wokół komponentu, można zmieniać wymiary, a niekiedy i orientację. Zakładka *Widget Properties* pozwala na wpisywanie parametrów zmieniających pozycję i wygląd ustawianego komponentu. Z lewej strony pulpitu znajduje się zakładka *Project Explorer*, na której w porządku wstawiania na ekran wyświetlone są nazwy użytych komponentów. Kliknięcie na nazwę wybiera komponent do edycji. Na dole usytuowana jest zakładka *Output Window*. Wyświetlane są na niej wywołania procedur i rozkazy niezbędne dla wyświetlenia na docelowym ekranie komponentów w takim kształcie i usytuowaniu jak w projekcie. Wybierając opcję *File* → *Export* → *Screen to DL* można zapisać w formacie pliku tekstowego wygenerowany szkielet wywołania procedur i rozkazów dla FT801. Podstawiając pod nazwy procedur ze szkieletu, procedury z biblioteki graficznej, można szybko stworzyć własny kod, który po wysłaniu do sterownika spowoduje wyświetlenie na ekranie wyświetlacza zaprojektowanych komponentów.

Przykładowe oprogramowanie, z którego można skorzystać ucząc się obsługi wyświetlacza, jest dostępne w materiałach dodatkowych na serwerze FTP.

Ryszard Szymaniak, EP



Rysunek 3. Okno główne programu Screen Designer

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: **ftp://ep.com.pl**, user: **11875**, pass: **6hhcxxtt**